## Práctica 1: Memoria compartida

## Sección 3: Cuda

El objetivo de esta sección es comprobar los beneficios de usar arquitecturas masivamente paralelas como pueden ser las GPUs con soporte para Cuda. En concreto, se paralelizará el método de multiplicación de matrices para poder usar las tarjetas gráficas del laboratorio, y se comprobará las mejoras obtenidas respecto de usar CPUs multicore. La práctica sólo se considerará APTA si es más rápida que las implementaciones CPU.

## Primera parte: Codificación

Como se observó en las implementaciones anteriores, cuando se trabaja con un número de cores reducido no es buena idea crear un número muy grande de Threads. Sin embargo, cuando se trabaja con arquitecturas con cientos/miles de cores, el objetivo es crear el número máximo de threads.

En este caso, el objetivo será crear un thread/kernel de cuda que multiplique una fila por una columna en paralelo. Para ello, se usarán las técnicas vistas en clase para crear varios bloques de threads de cuda. Se recomienda seguir el siguiente esquema de ejecución:

Inicio (main)

Leer matrices de disco duro

Copiar matrices a GPU y reservar espacio para el resultado

Lanzar N kernels de Cuda que realicen la multiplicación de una fila por una coumna (el antiguo método "MultiplicaVectores")

* + Calcular el número máximo de threads/kernels de GPU que se lanzarán (Filas\*Columnas)
  + Subdividir en bloques de ejecución bidimensionales de 32x32 threads (no se pueden crear bloques de más de 1024 threads)
  + Subdividir el número de bloques en grids bidimensionales. La dimensión en el eje "X" será de "Columnas /32", y en el eje Y será de "Filas/32". De esa manera se calculan cuantos bloques habrá que generar en función del número de filas y columnas.
  + Ejecutar los kernels en GPU. En cada kernel:
    - Averiguar la Fila y Columna asignada en función de los identificadores de bloque y thread
    - Multiplicar la fila /columna calculada, y guardar el resultado
  + Sincronizar la ejecución con la GPU
  + Traer los datos de GPU a CPU

Escribir resultados en disco

Aunque se deja libertad al alumno para poder implementar el kernel de CUDA como prefiera, se aconseja usar la siguiente cabecera:

**\_global\_ void multiplicaMatrices\_kernel( int numFilas, int numColumnas,**

**float\*\* matrizl , float\*\* matriz2, float\*\* resultado)**

**{**

**}**

## Segunda parte: Evaluación

En este caso se evaluará la mejora de la arquitectura GPU respecto de las implementaciones secuencial (una CPU) y paralela con reparto estático (8 threads). Se calcularán los tiempos de ejecución en GPU, y se calculará el speedup respecto de la versión secuencial, y se presentarán en las siguientes tablas:

Tiempos de ejecución:

Tiempos de ejecución:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Secuencial** | **8 Threads, División**  **estática** | **CUDA** |
| Tiempo (Segundos) |  |  |  |

Speedup respecto de la versión secuencial:

**8 Threads, División estática CUDA**

Speedup respecto de la división estática (solo un speedup):

**CUDA**

## Entrega de la práctica:

La fecha límite de entrega para poder optar a evaluación continua será el Lunes 1 de Mayol, a través del campus virtual. Ese día se entregarán todos los proyectos y código fuente generados en cada una de las tres secciones en las que se ha dividido la práctica. Igualmente se entregará una memoria con todos los datos de evaluación pedidos para cada una de las implementaciones. En las siguientes sesiones de laboratorio se demostrará el correcto funcionamiento de la aplicación, respondiendo correctamente a las preguntas que formule el profesor sobre la implementación.

En caso de no realizar la entrega, se perderá la evaluación continua y habrá que entregar la práctica el día del examen. En ese caso no